Evolutionary Robotics: From Simulation to the Real World using Anticipation

Cedric Hartland, Nicolas Bredeche

Evolutionary Robotics: From Simulation to the Real World using Anticipation

Cédric Hartland and Nicolas Bredeche

IA-TAO team - INRIA Futurs, LRI/CNRS
Bat.490, Université de Paris-Sud
F-91405 Orsay Cedex, France

Abstract. Evolutionary Robotics provide efficient tools and approach to address automatic design of controllers for automous mobile robots. However, the computational cost of the optimization process makes it difficult to evolve controllers directly into the real world. This paper addresses the key problem of tranferring into the real world a robotic controller that has been evolved in a robotic simulator. The approach presented here relies on the definition of an anticipation-enabled control architecture. The anticipation module is able to build a partial model of the simulated environment and, once in the real world, performs an error estimation of this model. This error can be reused so as to perform in-situ on-line adaptation of robot control. Experiments in simulation and real-world showed that an evolved robot is able to perform on-line recovery from several kind of locomotion perturbations.

1 Introduction

Evolutionary Robotics provide an approach towards building efficient controllers for autonomous (mobile) robots in the context of sparse, noisy and delayed rewards[1]. This is possible thanks to extensive use of evolutionary algorithm, a stochastic population-based optimisation process[3]. However, such optimization algorithms require high computational resources. As a consequence, optimization is usually performed in a simulated environment and often end up with behaviors that cannot be implemented into an autonomous robot in the real world.

In this paper, we address the key problem of behavior tranfer from simulation to the real world in the context of a task-optimized mobile autonomous robot, sometimes refered to as the *reality gap*[**?**]. More precisely, we are concerned with adaptive on-line calibration process wrt. locomotion perturbations : calibration issues, partial hardware and/or software failure, medium and long lasting changes in the environment, etc.

Our approach relies in the use of an anticipation mechanism that makes it possible to capture specificities of the simulated world where optimization took place. The anticipation mechanism is then able to quantize the actual locomotion error in the real world with regards to intended motor outputs from the controller. It is then possible to use this error signal in order to perform on-line adaptation to recover from several kind of motor calibration error and/or medium- and long-lasting motor noise.

In the following section, we present the problem setting and review the litterature for possible candidate solutions. Next, we describe an approach based on training an anticipation mechanisms in order to modelize the simulated environment and, in a later step, to approximate the error of this model with regard to robot's behavior in the real world. This mechanism makes it possible to adaptively correct motor outputs so as to comply with the robot physical dynamics in the real world. In section 4, our approach is experimentaly validated for an evolved wall-avoidance behavior and several experiments show the robustness towards different kind of locomotion perturbations.

2 Problem setting and related works

The Evolutionary Robotics [1] approach addresses the problem of automatic design of robotic controllers by relying on population-based stochastic optimisation algorithms, i.e. evolutionary algorithms. Such algorithms are particularly well fitted when the objective function (i.e. the task) is difficult to describe. These stochastic optimisation algorithms perform on a generational basis (i.e. optimisation at step i depends on step $i-1$) and rely on the exploration of the space of possible solutions through a population of candidate solutions by combining selection operators (most fitted candidates are likely to survive from one generation to another) and ideally well-suited variation operators (candidates may be recombined and/or altered to diffuse supposedly good characteristics as well as to efficiently explore the search space). These algorithms have been shown to be very efficient and to achieve human-competitive results on numerous problems where standard learning algorithm are difficult to apply, which is a key advantage in robotics.

However, controller optimization has long been rightfully criticized because of a poor modelization of the robot physical behavior in the real world. Short and long-term perturbations are either ignored or loosely modelized within the simulator because of the task extreme difficulty[6, **?**]. Hence, evolved controllers are very difficult to implement in a real world robot and experiments are usually limited to simulated environments.

In the scope of this paper, we are concerned with locomotion perturbations in the real world. In this context, it is possible to identify several kind of perturbations, most of them (except no.2) resulting in a stable or changing directional bias during locomotion:

1. **calibration perturbation** depending on wheels or motor characteristics at initialization (diameter, power, etc.);
2. **punctual perturbation** that occurs more or less frequently, but are always very limited in time (e.g. slip/sliding, collision, etc.);
3. **long-lasting change** due to a partial motor failure (e.g. minor hardware failure, lasting energy supply problem, external perturbation, change in the wheel texture, etc.);
4. **intensity-changing perturbation** (e.g. ongoing loss of energy for one motor, agravating motor problem, etc.).

Among these possible perturbations, it should me noted that the second one is usually easy to handle by controllers thanks to immediate recovery and does not need adaptation (i.e. a single motor command towards the other direction may compensate sliding). However, all other perturbations cannot be adressed in such a way since non-adaptive controllers would oscillate between goal-oriented control and error compensation (see section 4 for an experimental illustration).

One extreme approach is to evolve controllers directly on real robots. While this makes it possible to completely avoid the problem at hand, it is extremely time consuming both from the robot viewpoint (for obvious reason, evolution in the real world is **much** slower than in a simulated environment) and from the human supervisor viewpoint (who has to deal with everything from minor annoyances, major software/hardware failures and coffee breaks). For example, the experiment described in [7] took more than 27 hours to perform a single evolution - note that a similar evolution could be performed in less than twenty minutes on today Personal Computers.

In a slightly different setup, some works have adressed the problem of adaptiveness to lasting changes in the scope of evolutionary robotics. The work presented in [4] addresses the problem of evolving an autonomous mobile robot controller for exploring a simple arena. During evolution, environmental variation is modified inbetween each generation ("night" and "day" alternate) and leads to a change in the IR proximity sensors behavior. The controller is a neural network with four inputs (IR sensors) and four outputs (two motor outpus, and two "models"). In fact, this network is divided in two parts: (1) the control network produces the motor output and (2) the model network (an oracle) produces the *desired* motor outputs. All weights are evolved, but during evaluation, control network weights are adjusted (with standard back-propagation) using the model network output as a reinforcement signal. The authors shows that such a network makes it possible to quickly adapt to the previously mentionned perturbations.

However, recent works in [5] show that this approach is not robust as soon as the robot is runned longer than the time used for evaluation during evolution (i.e. control network converge towards model network). The authors propose a new architecture, based on a neural network controller that embeds a control network (as previously) and an anticipatory network. In this case, they show that robustness in the long term is possible for the same "night" and "day" experimental setting. In this case, the anticipatory network predicted sensor inputs at $t + 1$.

While the cited works where concerned with adaptation towards long-lasting sensor perturbations, the anticipatory mechanism provide an efficient way to take into consideration environmental perturbations. Indeed, the main point of such a mechanism is to provide an error rate of the awaited consequence of an action on the environment. In the next section, we show how to exploit this error signal to address the problem of motor perturbations.

3 Approach

In the scope of evolving a controller within a simulated environment, we propose to build a partial model of this very simulated environment. The goal of such a model is to be able to provide a basis for detecting anomalies (i.e. perturbations) once the robot is immersed in the real world. That is, we intend to build an error detector rather than an exact model of the world. Biologists have long been aware of the role of anticipation in animals and the ability of such a module to perform agent and/or world behavior error detection. Formally, the anticipation function ($F_{anticipation}$) is defined as:

$\forall t \in T$, $F_{anticipation}(\omega(state_t)) \rightarrow \theta(state_{t+1})$

With $state_*$ defining all possible information that characterize the robot in its environment (e.g. sensory inputs, action outputs, proprioceptive information, memory, etc.) and with ω and θ as functions that perform a selection on a subpart of all available information in a given *state* of the robot in the environment (e.g. only IR sensors as inputs and only motor as outputs).

In the next two sub-sections, we shall define (1) the anticipation module that predicts the awaited variation between actual and future state and (2) the correction module that corrects desired actions so as to cope with possible perturbations.

3.1 The Anticipation Module

In a previous section, we showed previous work that relied on sensory information anticipation[5]. However, locomotion is mostly independent of the sensors (as far as no obstacle comes into contact). The question is: how to measure perturbation during locomotion? A relevant approach is to rely on the variation (or difference) between the awaited consequences of an action and the actual observed consequences. The next question is to identify the relevant sensors from which to measure such a variation. For example, a sonar belt is hardly fitted to do this (perceptual aliasing, very noisy - it is mostly targeted at low-level object avoidance). However, many sensors provide evaluation of *relative* movement in the environment: compass, optical flow, inertial gyro, external-source odometer (e.g. LED-LDR circuit), etc. For all these sensors, the anticipation module is focused on predicting variations between actual and future sensor values (e.g. compass variation in radians).

Figure 1 shows our architecture. The anticipation behavior predicts the variation in orientation from desired motor outputs. At the next step, this variation is then confronted with the actual observed variation in orientation so as to evaluate a prediction error, which is finally used to correct and convert *desired* outputs into *effective* outputs. This architecture shows two very interesting properties:

1. the anticipation module is completely independent from other modules and may be trained separately (e.g. using a simple wander behavior) or in par-

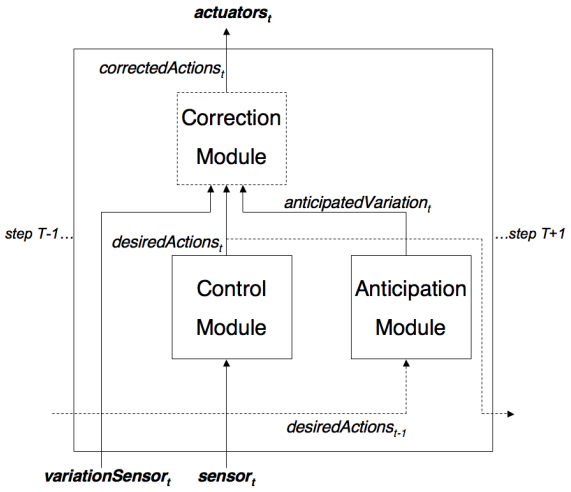

Fig. 1. Overview of the control architecture

rallel with another learning task (e.g. behavior optimization)[1];
2. training can be easily performed as a supervised learning task (regression) where each step provides the source data from learning step t and the oracle output for learning step $t-1$ ($\forall t$), that is : $\forall t \in T$, find $F_{anticipation}$ defined as $F_{anticipation}(motor_t) \rightarrow variation_{t+1}$. In section 4, we will show that this training can be easily and quickly achieved using a very simple neural network predicting compass variation.

3.2 The Correction Module

According to figure 1, the anticipation signal is used as a source to correct the desired motor commands, i.e. efficient commands in a simulated environment. Of course, this module is added to the architecture after evolution, i.e. only when the robot is immersed in the real world. In the following, we are concerned with a two-wheels khepera robot but it should be noted that extension towards diverse hardware is rather straight-forward. The following algorithm describes the correction module :

1. $variationAmplitude = abs(difference(variationSensor_{value}, anticipatedVariation_{value}))$
2. $variationDirection = sign(difference(variationSensor_{value}, anticipatedVariation_{value}))$
3. if ($variationDirection > 0$) then $\alpha := \alpha * (adaptiveSpeed * \sigma(variationAmplitude))$
4. if ($variationDirection < 0$) then $\alpha := max(1, \alpha/(adaptiveSpeed * \sigma(variationAmplitude)))$
5. $correctedMotor_{right} := desiredMotor_{right} * \alpha$
6. $correcterMotor_{left} := desiredMotor_{left} * (1 - \alpha)$

[1] Training provides a much faster and efficient way to build the anticipation module compared to manual definition by the supervisor.

α is the corrective value and *adaptiveSpeed* is the speed at which α should be adjusted whenever a prediction error occurs[2]. Moreover, *adaptiveSpeed* is multiplied by the normalised and bounded *variationAmplitude* (thanks to function σ - maximum threshold is set to avoid correction oscillations). All values are defined between 0 and 1.

The above algorithm means that the Correction Module performs on-line adaptation based on the difference between predicted motor consequences and actual observed motor consequences on the position of the robot in the real world. Long-term adaptation is performed by gradually modifying the α term. This term is used to adjust motor calibration and is constantly converging toward the (possibly changing through time) optimal value[3].

In the next section, we show several experiments in simulation and real world that illustrate our approach.

4 Experiments

4.1 Implementation Issues

The following experiments were performed the SIMBAD robotic simulator developped in our lab and a Khepera mobile robot with color camera.

SIMBAD is an open source Java 3d robot simulator for scientific and educational purposes[9][4]. It provides a simple basis for studying Situated Artificial Intelligence, Machine Learning, and more generally AI algorithms, in the context of Autonomous Robotics and Autonomous Agents. Moreover, it embeds PICONODE, a Neural Network library (feed-forward NN, recurrent NN, etc.) and can be easily used in conjunction with any Evolutionary Algorithm library[5]. We rely on Sean Luke's ECJ[10], a powerful Evolutionary Computation library in Java, for the evolutionary part.

The robot used is an extended khepera[6] with the wireless colour camera, 8 infra-red sensors and 2 motors (i.e. two wheels). The SIMBAD simulator provides a simulated khepera and control architecture can be switched from simulated to real world. The robot used in simulation slightly differs from the real one. The variation sensor for the simulated robot is provided by a denoised compass while it is provided through a simple visual tracking algorithm for the real robot. This algorithm tracks specific landmarks in the environment and compute orientation variation from the landmark's movement.

[2] in this case, we assume that *variationDirection* is positive (negative) if the robot is biased towards going right (left).

[3] conergence is guaranteed by the threshold in the σ function.

[4] The Simbad Package is available from http://simbad.sourceforge.net/ under the conditions of the GPL (GNU General Public Licence).

[5] Recent versions do indeed provide such a library.

[6] http://www.k-team.com/

4.2 Experimental setup

So as to provide a basis for the evaluation of our architecture, we start with evolving a controller for a given task and training the anticipation network in the simulator. On the previously shown figure 1, this means we are successively concerned with the *control module* and the *anticipation module*.

Evolving the Control Module :

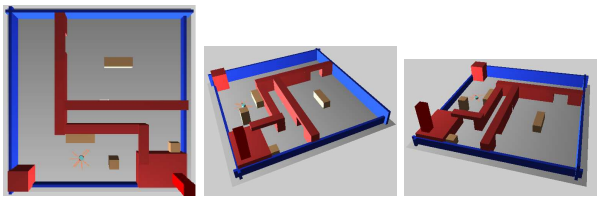

Fig. 2. Evolution environment for the controll

The task to be optimized is defined as visiting the maximum number of places in the environment shown in figure 2. The *control module* is a multi-layered perceptron. The perceptron have 8 inputs (one for each IR sensor on the khepera robot) and 2 outputs corresponding to the motor outputs (left an right motor). The evolution optimize the weights on the connections between neurons. Problem properties and parameters are defined as:

- 15 neurons (8 inputs, 4 hidden, 2 outputs, 1 bias);
- the network is fully connected (46 weights);
- 100 individuals, (20+80)-ES, mutation rate is 0.2;
- environment is divided into $i * j$ zones to be visited;
- fitness :

  Optimize F such that $F = \sum_{i=0}^{n}(\sum_{j=0}^{n}(explored_{i,j}))$
  duration: T_{max}.
  $explored_{i,j}$ is 1 if explored, 0 if not.
  It stops right after a collision (implicit penalization).

Results are shown on figure 3: the robot quickly succeeds in exploring all places.

Training the Anticipation Module :

As seen in the previous section, the *anticipation module* can be trained using desired motor commands at time t as input and observed orientation variation at time $t + 1$ as output. The goal is to predict as accurately as possible the future state of the robot in the world, and then compare it to the effective state of the world. The error rate of the anticipation mechanism should be low in the simulated environment, and unknown at first in the conditions of a real

Fig. 3. Fitness function for the control

environment. In this case, two situations arise: either it is low (simulation was accurate), or it is high, allowing to detect noise and perturbation and enabling the *correction module* to act accordingly (see previous section).

This is indeed a simple straight-forward regression learning task ($R^n \rightarrow R^1$). In order to perform this regression task, the anticipation mechanism is implemented as a multi-layer perceptron with 2 inputs (motor commands) and one output (orientation variation). Weights between neurons are learned using the standard back-propagation learning algorithm.

First, 1.000.000 examples are extracted from random wander controls of the robot in the simulator. Then, learning is performed on a subset of the examples and tested with another subset - despite the great number of examples, learning is always performed in less than 10 seconds. Learning is repeated 10 times to get a good accuracy of learning performance. Several topologies were tested, from 0 hidden nodes to 20 hidden nodes (not shown here) - best results are archieved by the network with no hidden layer (see fig. 4), which is not really surprising.

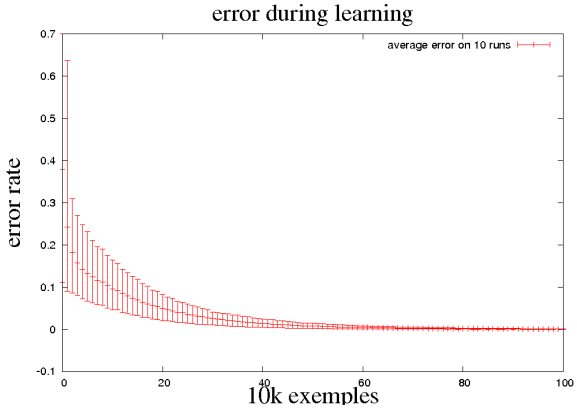

Fig. 4. Error rate during the learning

Adding the Correction Module :

Once both modules are completed, we add the *correction module* and put our architecture to the test. This means that no evolution/learning take place anymore, only adaptation occur in the *correction module*. In the following, we present four experiments:

- On-line calibration correction
- On-line adaptation to continuous wear
- On-line adaptation to changing environmental noise
- On-line adaptation for a real-world robot

The three first experiments are performed in simulation: different kind of perturbations are added in the simulator *after* evolution is completed and the adaptation ability is evaluated. The orientation sensor is based on an on-board compass. The final experiment is performed using a real Khepera robot and on-line adaptation is evaluated for a simple go-forward behavior. In this case, change of orientations are detected using visual tracking of a landmark - this results in the same architecture, only Khepera robot do not provide compass. It is important to note that from the controler viewpoint, the type of value is interpreted in the exact same fashion (i.e. a change of orientation). As a consequence, an evolved robot using compass can be implemented straight-forward in a real robot using visual tracking as long as normalization is the same.

In all the following experiments, robot locomotion traces are shown along with the prediction error - in the case of non-anticipation-based correction controllers, anticipation is nevertheless computed, albeit not used, so as to plot the prediction error.

4.3 On-line calibration correction

The first problem to address in a real environment deals the initial calibration of the robot. We evaluate it in the case (1) of a hand-written go-forward controller and (2) of the evolved controller. Figures 5 and 6 respectively show the results for both setting. In both cases, the anticipation module provide a clear correction and control quickly converge to the awaited behavior. Then again, the figures show that anticipation error decrease over time as adaptation is performed. In the case of the evolved controller, error peaks occur when avoiding a wall.

4.4 On-line adaptation to continuous wear

This second experiment addresses the problem of continuous wear: a wheel which diameter or motor power decreases over time. In this setup, perturbation intensity increases in the form of a logarithmic function over time, until a threshold is reached. The correction mechanism is exactly the same as before and same tests are performed (hand-written and evolved controllers). For both experiments, the error is quite important at first, and the anticipation-based correction quickly correct accordingly and keeps on correcting to maintain the prediction as low as possible. Results are shown in figures 7 and 8. Note that correction for the evolved behavior makes it possible to avoid a crash (see fig.).

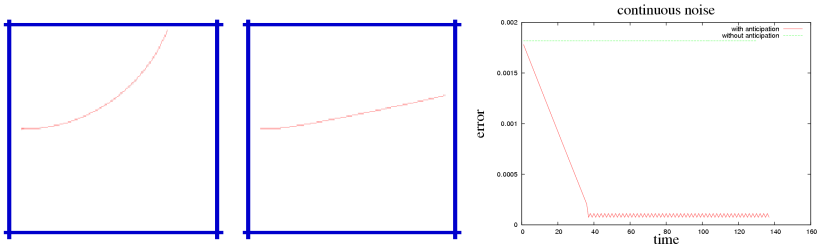

Fig. 5. go-forward behavior. left: without correction, middle: with correction, right: prediction error

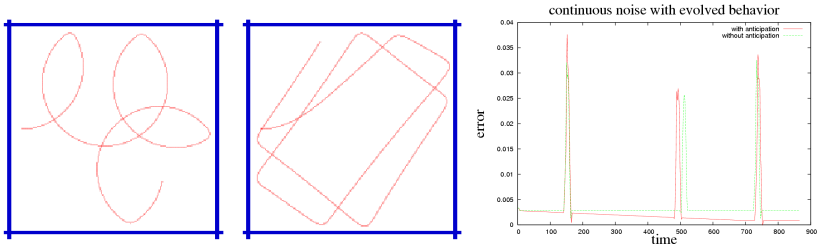

Fig. 6. evolved behavior. left: without correction, middle: with correction, right: prediction error

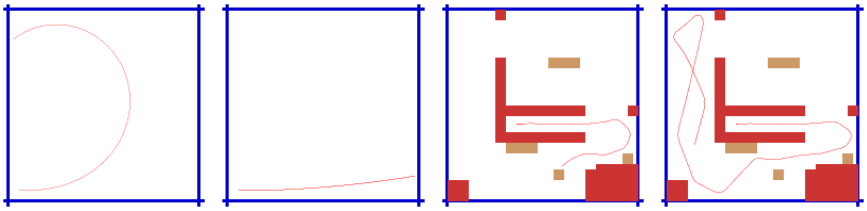

Fig. 7. behaviors for continuous wear. left: ad-hoc without correction, middle-left: ad-hoc with correction, middle-right: evolved without correction, right: evolved with correction

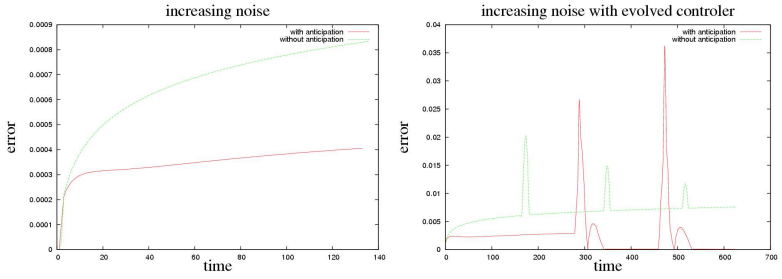

Fig. 8. prediction error for continuous wear. left: ad-hoc straight-forward behavior, right: evolved behavior.

4.5 On-line adaptation to changing environmental noise

In this third experiment, we are concerned with the robustness towards perturbation changing through time (e.g. non-homogeneous grip due to environmental changes). In order to achieve this, we increase and decrease continuously the perturbation on both the wheels, in a form of a sinusoidal noise function. Results in fig. 9 show clearly that the anticipation based mechanism allows to correct and reduce significantly the perturbation on the wheels while without correction, the robot deviates more and more from its initial destination.

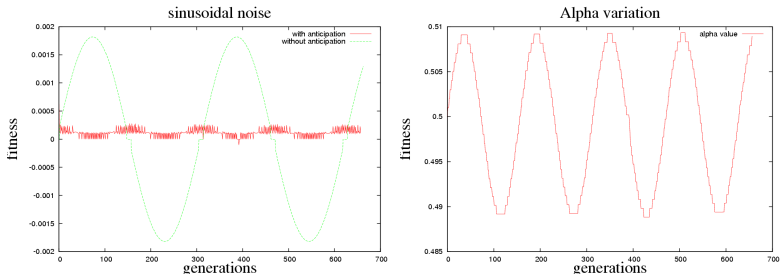

Fig. 9. Left: prediction error for changing environmental noise. Right: alpha correcting value is adapted over time.

4.6 On-line adaptation for a real-world robot

In this final experiment, we reproduce the same setup as before on a real Khepera robot with a 2D camera (see section 4.1). Perturbation is that of the real world, the controller is a simple go-forward ad hoc behavior and the *Anticipation Module* is trained as before. The orientation sensor is implemented using a visual tracking algorithm which track a red landmark in the environment. The robot is placed in straight line in front of the red landmark. Figure 11 respectively show the behavior of the robot without and with anticipation-based correction. In order to compare the two settings, we compute the cumulative prediction error over time that is given by the anticipation module in both case (see fig. 10 - in the setup where no anticipation-based correction is done, anticipation is computed to plot the prediction error but not used). In this figure, it is clear that the anticipation and correction modules is able to reduce prediction error.

5 Conclusion

In this paper, we have adressed the difficult problem of transferring a simulation-based evolved controller to an autonomous mobile robot in the real world. This is a key problem in Evolutionary Robotics since evolved controllers are rarely

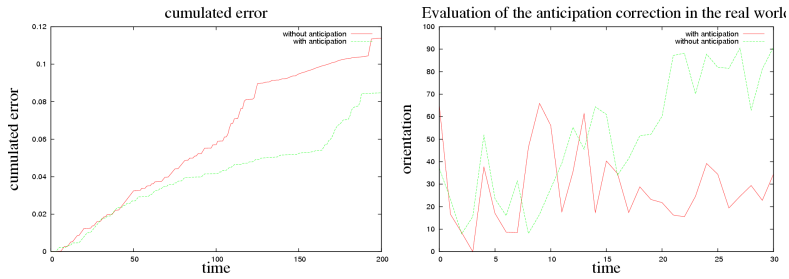

Fig. 10. Real robot error rates. On the left, cumulative error, and on the right, orientation toward the target. The target is in centered on approximately 30-40

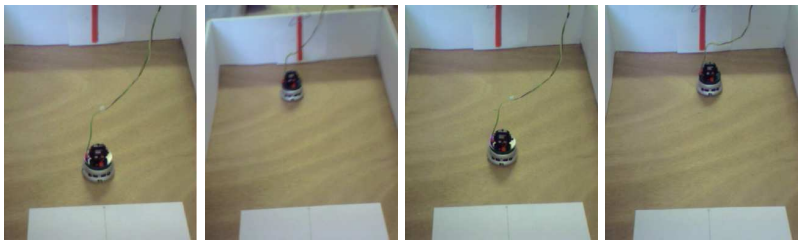

Fig. 11. 2 left images: go-forward without correction - 2 right images: go-forward with correction

used in real robots because of the difficult task of exactly simulating what is relevant from the real world.

The main contribution of this paper is to propose a new generic control architecture for Evolutionary Robotics that relies on an anticipation module so as to perform on-line adaptation towards locomotion perturbations. This module is learned during evolution and then used during real-world operation to correct output commands from the task-oriented evolved controller.

Experiments showed that this anticipation module is able to capture part of the simulated world model and makes it possible to compute a prediction error between awaited and actual consequences of an action in the real world. This prediction error was shown as a key feature for on-line adaptation towards several kind of perturbations and failures.

Future works include more real-world studies of the adaptive capacity of the proposed architecture in the scope of using more generic and reliable orientation variation sensors such as optical flow based sensor rather than compass or visual tracking sensor. We also intend to extend our approach to the case of more complex robotic systems such as legged robot, where such anticipation module may be used for failure detection and recovery.

Acknowledgment

This work was supported in part by the PASCAL Network of Excellence and by a CNRS TCAN grant (Nerobot project).

References

1. S. Nolfi and D. Floreano. Evolutionary Robotics. MIT Press, 2000.
2. R. Brooks. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, Vol. 2, No. 1, pp.14-23 1986.
3. D. Goldberg. Genetic Algorithms in search. Addison-Wesley, 1989.
4. S. Nolfi and D. Parisi. Auto-teaching : networks that develop their own teaching input. Proceedings of the second ECAL, 1993.
5. N. Godzik, M. Schoenauer and M. Sebag. Robustness in the long run : Auto-teaching vs. Anticipation in Evolutionary Robotics. Proceedings of PPSN, 2004.
6. U. Nehmzow. Quantitative analysis of robot-environment interaction - on the difference between simulations and the real thing. In Proceedings of Eurobot, 2001.
7. O. Miglino, S. Nolfi, H. Hautop Lund. Evolving mobile robots in Simulated and real environments. Artificial Life 2 (417-434) 1995
8. N. Jakobi, P. Husbands, I. Harvey. Noise and the reality gap : the use of simulation in evolutionnary robotics. Lecture Notes in Computer Science 1995
9. L. Hugues, N. Bredeche. Simbad : an Autonomous Robot Simulation Package for Education and Research. Accepted for publication at The Ninth International Conference on the Simulation of Adaptive Behavior (SAB'06), Roma, Italy.
10. S. Luke. ECJ: A java-based evolutionary computation and genetic programming system, 2002. http://cs.gmu.edu/∼eclab/projects/ecj/.